GPS Integrity Monitoring

Tom Temple*

May 10, 2005

Abstract

This paper seeks to explore the potential of evolutionary algorithms for determining hard error bounds required for the use of GPS in safety-of-life circumstances.

1 Introduction

The Global Positioning System (GPS) has proved to be very useful in a number of applications. Aircraft navigation is among the most important. But in the safety-of-life circumstance of precision approach and landing, we do not have sufficient guarantees of accuracy. While differential GPS (DGPS) is precise enough much of the time, a controller needs to be able to put strict error bounds on the position estimate in order to be able to use DGPS for precision approach or landing. This paper explores the use of an evolutionary algorithm to determine these error bars.

2 Problem Specification

Rather than solve the "Is it safe to land, or not?" question, I will attempt to answer a slightly more general problem, "What is the worst

*Lincoln Laboratory

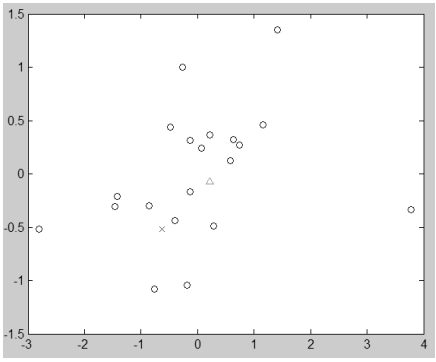

Figure 1: The 2-dimensional projection of the distribution of positions calculated from subsets of the pseudo-ranges. The green triangle is the estimate using all of the pseudo-ranges and the red x is the true position. The axis are in meters.

that the error could be?" Given a set of satellite ranges, we would like to estimate an upper bound on position error. We are allowed to underestimate the bound no more than once in 10 million trials. Simultaneously, one would want the service to be available as much of the time as possible. In other words, we would like to keep overestimation to a minimum while maintaining the strict rule on underestimation.

As the number of satellite ranges available increases, the set of equations that the receiver must solve become increasingly over-specified. The goal is to use this over-specification to generate a number of partially independent estimates of position. The distribution of these estimates can be used to estimate the distribution from which the position measurement (using all satellites) was selected.

Figure 1 shows the two-dimensional projection of the distribution of twenty subset positions. As one might expect, the true position lies within this distribution. Given such a distribution, the goal is to draw the smallest circle that surely contains the true position.

3 Previous Work

I am primarily building on work by Misra and Bednarz[3]. They proposed an algorithm, called `LGG1`, which consists of three elements.

- A method of selecting subsets of satellites with good geometries
- A characterization of the distribution of subset positions
- And a "rule" function that turns this distribution into an error-bound.

They demonstrated that such an algorithm, if sufficiently conservative, could give an error bound that was sufficiently stringent. If the rule function was a linear function, the error bound was sufficiently stringent regardless of the underlying range error distribution. The recent subject of my research has been testing and improving the algorithm.

I retain all three elements of `LGG1` algorithm but will change each of them. For this work I will be focusing on determining the rule function. I will explore using an evolutionary algorithm to determine a rule function that exploits more expressive characterizations of the subset position distributions.

The original algorithm used a single metric to quantify the distribution of subset positions. The number that it used was the distance between the furthest two positions and they called it "scatter." The rule was a constant times this scatter. Such a rule could be found quickly given a dataset with millions of error–scatter pairs.

4 Current Extention

The current work seeks to explore using a more expressive description of the subset position distribution. We would like to utilize the fact that there is much more information in the distribution than the scatter metric. So rather than characterize the distribution with one

summary number, I use a set of what I am going to call "p-scatters" computed as follows.

$$S_p = \left(\frac{\sum |x - \hat{x}|^p}{n} \right)^{1/p}$$

You will note that S_2 is the standard deviation and S_∞ is similar to the metric proposed by Misra. A value p need not be an integer nor be positive. In this paper, the values of p that are used are chosen by hand. Choosing the "best" values of p is a problem with which I am still grappling. Future work will tell whether a learning algorithm can be applied to determining which values are the most telling of the distribution.

One property desirable in a rule function is that it be scale independent. If all the errors double, the scatter should double and the error bound should also double. A quick check reveals that the p-scatters have this property. So will the rule function as long as it is a linear combination of the p-scatters. This means our rule function can be represented as a vector of weights. We have reduced the problem to a parameter estimation problem.

4.1 Parameter Estimation

The main idea is that the subset position distribution is somehow similar to the distribution from which the estimate is sampled. If we can determine how the distributions are related, we can use the position distribution to estimate the error distribution and determine an error bound. If the errors were truly Gaussian, we could simply estimate the standard deviation, S_2, and simply multiply it by six and call it a day. Alas, they are not and it will not be so easy.

In general, parameter estimation problems consist of finding "optimal" values for the parameters. "Optimal" values can be defined as those that maximize or minimize some function, for instance, minimize the squared error, or maximize the margin of classification. The sense of optimal in this case is more tricky. Our value function on (error estimate) error is very non-linear. To the negative side of zero

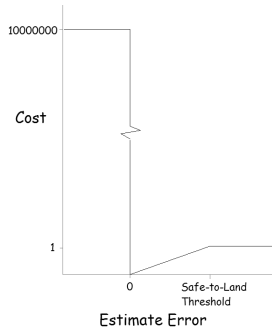

Figure 2: The cost associated with making errors in our error-boundestimate.

it has a value that is *10 million times* the value on the positive side (see Figure 2).

Given a set of p-scatters, consider the space of p-scatters with one additional dimension, namely error length, as the "vertical" axis. Now imagine that we plot 10 million data points in this space. The current problem is the same as finding the best hyperplane such that all the points fall bellow this plane. If we demand that this hyperplane, with normal $\hat{n}$ pass through the origin (as we must if our rule is going to be linear) then we could define "best" to mean the one for which

$$\sum_{d \in \text{scatter data}} -\hat{n} \cdot d$$

is minimized. In high dimensional scatter-spaces and with millions of data points, there is no tractable way to find such a plane analytically.

This was the problem that I was pondering when Justin presented learning the parameters of the chess evaluation function by means of an evolutionary algorithm[1]. The parameters in chess are not easily defined by a optimization problem. You want parameters that maximize the probability of winning the chess game! Just like in my problem, the algorithm wants to minimize a function that it cannot simply compute. If an evolutionary algorithm could find the chess values, I decided it would be worth a try on my scatter weights.

5 Evolutionary Algorithm

5.1 Simulation Framework

For other facets of my research, I had already developed a simulation framework that models a GPS constellation and receiver. The simulation can provide pseudo-range snapshots for given times and locations on the earth. These pseudo-ranges are selected from a pre-determined error distribution. The position error is the distance between the true position and the one computed by the modeled receiver. The subset distribution is generated by giving the receiver 20 subsets of the pseudo-ranges. The run-time of the simulation is linear in the number of subsets. For the current work, 20 seemed sufficient to estimate scatter while not being taking overly long. The quality of a subset is determined by its Dilution of Precision (DOP) which is a function of its geometry. I defined a cutoff DOP of 3 above which subsets are not considered. [1]

5.2 Population Details

I used a population of 100 individuals although the computational expense of increasing this number is relatively minor. An individual in the population consists of a vector of weights, θ. A trial consists of simulating a pseudo-range snapshot and computing the scatter metrics, ϕ. Then for each member of the population we compute $\phi^T\theta = \widehat{\varepsilon_{\max}}$, the estimated maximum error. If $\widehat{\varepsilon_{\max}}$ is less than the true error, the individual is killed. If $\widehat{\varepsilon_{\max}}$ is less than the threshold for a safe landing (10 meters in this case), the individual is given a point. On an intuitive note, these cases can be considered to be crashes and successful landing attempts, respectively.[2] All of the remaining cases (*i.e.* when $\widehat{\varepsilon_{\max}}$ is greater than both the true error and the safety threshold) are

[1] The constellation consists of 48 satellites in 6 orbital planes which, to a receiver, is similar to the one we can expect to see when GPS and Galileo are simultaneously operational. The receivers are always between the Arctic and Antarctic circles. The pseudo-range error distribution is a mixture of Gaussian $.9\mathcal{N}(0, 1m) + .1\mathcal{N}(0, 3m)$ which is roughly ten times the range error typically exhibited by DGPS.

[2] Technically, crashes are only cases when the estimate is bellow the safety threshold while the error is not. However with different thresholds or error scaling, any under-

those cases when a landing was not attempted. Hence, the number of points that an individual has is equal to the number of successful landings they have had.

A generation consists of 100 such trials. At the end of a generation, the individuals who are still living create progeny in proportion to the number of points that they have accrued. The children are created by mating or by mutation. Mating consists of a random blending of the weights of the parents while mutation consists of adding random variation to the parent. The decision of whether to mate or mutate is made randomly for each child. As pointed out by Jeremy and Justin[2], the proportions (1:1 in this case) were selected arbitrarily. This is a point where "meta-evolution" could potentially reap dividends.

5.3 Convergence

There was a serious problem with this setup. There is not enough experience in 100 trials to weed out the overly aggressive individuals. This means that the population would be dominated by the most aggressive pilots and then be wiped out when a difficult case came along. We would need on the order of 10 million trials to be confident that we had exposed them to sufficient variation. But generating 10 million new trials for each generation would take far too long. To fix this problem, before seeing the 100 new cases on which virility would be based, first an individual would have to survive a set of torture tests drawn from the most difficult trials seen previously. Initially, this set was filled with 100,000 ordinary trials but then they were gradually replaced by the most difficult ones encountered during evolution. In addition to solving the stated problem, it also ensured that future generations would be able to survive all of the hard trials that earlier generations had seen. In other words, without this addition, generations might forget how to deal with cases that their ancestors had survived.

In the lecture given by Jeremy, Justin and Jennifer, they described

estimate could result in a crash. Therefore all such instances are considered to be crashes.

reducing the variation inherent in mutation over the course of evolution to ensure convergence[1]. While this ensured convergence, it did not ensure a satisfactory convergence. If the population converged too far, it tended to be wiped out by rare (once every 1,000 generations) but very difficult new cases.

The first fix for this was to keep an evolution history that allowed de-evolution. A difficult case might require that the entire population de-evolve a long way before there were members who could survive. This typically would only be a very small number of individuals. With so few members, there was very little variation as well as no way of comparing fitness. In other words the sole survivor might be the sort who *never* attempts a landing.

In the end, I decided to avoid convergence. Instead, I tried to maintain the variance of the population at a constant level. This is accomplished by having the frequency and magnitude of mutations stay constant. The problem with that is that there is no time when the evolution appears done. Another problem is that the evolution might be unable to settle down into a narrow minima. The benefit is that there is always substantial variation in the population. It is very seldom that it is forced to de-evolve. Similarly, if the population is very badly thinned, the variation will quickly return.

6 Experimental Results

Since a simulation to adequately demonstrate the required reliability of 1 error per 10^7 trials takes about a week, I did not have time to run at that level of accuracy. Instead I used a reliability on the order or 1 per 10^5. The number of trials per generation was set at 100 with the lower reliability in mind and will likely be increased for longer simulations. Each evolution consisted of 1500 generations. Then they were tested against a test suite consisting of 10^5 trials. As of yet, there is no rigor in determining these numbers. Rather, they were chosen to be large enough to be meaningful while small enough to allow me to run a reasonable number of cases.

Rule functions (so long as they provide satisfactory safety) are evaluated in terms of the fraction of the time that they attempt landings, Availability of Service (AS). A landing is attempted if the rule function, $\phi^T\theta$ is less than the safety threshold of 10m.

For each evolution, the 5 most healthy individuals at the end of the evolution are reported. In Table 1, I have shown results for evolution using $p = [1, 2, \infty]$. The first two are constrained so that all the weights in θ must be non-negative. The third example is not constrained. Table 2 shows the results from running the evolution with other sets of p-scatters.

For comparison, I am also presenting the AS for the `LGG1` algorithm as well as the tightest single-scatter rules possible. Note that the tight rules are generated on the *test* data rather than the training data, therefore they are tighter than could have been learned. Also included are the results from minimizing the Perceptron Criterion function. The details of that methodology are beyond the scope of this paper. Suffice it to say that those results are generated by very patient gradient descent in [error rate, θ] space.

7 Conclusions and Future work

While I would say that the evolution seemed successful, the results were generally disappointing. Table 1 shows that in the three-dimensional case, the evolutionary algorithm can generate a rule function of quality very near that of the other methods available. That also means that evolution failed to distinguish itself as a particularly good way to determine these parameters. This could be due to the fact that the variance is kept relatively high during the entire evolution. That could mean that the population could "vibrate out" of narrow minima.

The results for higher dimensional parameter spaces were unexpectedly bad. Since these spaces have the earlier case as a subspace, one would expect the results to be at least as good. The difficulty was

Test	Weight			AS
	p=1	p=2	p=∞	
Evolution 1	1.1832	2.5478	1.4925	0.3070
	1.2236	2.5292	1.4912	0.3045
	1.1606	2.5472	1.5200	0.3016
	1.1991	2.5537	1.5005	0.3015
	1.2563	2.5321	1.4899	0.3000
Evolution 2	1.1897	2.4237	1.5522	0.3068
	1.0256	2.5540	1.5642	0.3056
	1.8587	2.5982	1.1938	0.2977
	1.0494	2.6369	1.5486	0.2945
	1.0274	2.5322	1.6210	0.2919
Evolution 3	9.7900	-0.8000	-0.6800	0.3330
	7.8352	0.7793	-0.5926	0.3245
	6.2668	-0.7404	0.9330	0.2908
	6.0099	0.6367	0.4130	0.2753
	6.1092	-0.7742	1.1085	0.2633
Tightest fit	7.2766	0	0	0.3325
	0	6.3824	0	0.3398
	0	0	3.7250	0.2164
`LGG1`	0	0	4.5	0.1004
Perceptron	1.8256	1.7893	1.5838	0.3081
2.5281	2.1994	1.0281	0.3188	
2.5166	2.1739	1.0463	0.3187	

Table 1: Learned weights and Availability of Service (AS)

p	Trial 1 weight	Trial 2 weight	Trial 3 weight	Trial 4 weight
.001	x	x	x	.8462
.01	x	x	2.7628	.5965
.1	x	17.0594	-1.9202	2.6827
.5	x	11.3105	4.9851	3.5754
1	2.7564	-2.8706	-4.9647	.2209
1.5	x	-7.6041	5.6760	-2.2266
2	.6937	-9.3782	5.6863	.2611
3	6.6544	x	x	2.7220
4	3.7766	x	x	x
∞	-3.9807	1.8966	-2.1745	.7474
AS	.2673	.2358	.2707	.4196

Table 2: Weights and Availability of Service using other p-scatters

probably due to the higher prevalence of local minima in the higher dimensional spaces. But then the fourth trial, with 9-dimensional scatter, had very good results. I will explore whether that was due to over-fitting or not. If they are valid, then the evolutionary algorithm might be the only way to estimate the parameters in such a high dimensional space. Another disappointment is that when the set of p-scatters is changed, the weights can change sign, and important values can suddenly become unimportant. This means choosing a good set of p values isn't simply a matter of finding the most informative individual values and collecting them.

The main difficulty then seems to be choosing a good set of p values with which to compute scatters. Picking too many parameters adds more local minima to our parameter space and creates a risk of over-fitting. Too few under-utilizes the information in the position distribution. I will try to extend the learning algorithm to also include picking which p-scatters to consider.

And of finally, before I approach the FAA with any of this, I need to collect more data... more data in terms of more validation cases,

larger simulations, larger subsets and various satellite geometries. I need to test the algorithm against more pathological error models. And, needless to say, I must be more rigorous.

References

[1] Fox, Novosad, and Pouly. Cognitive game theory. Lecture, April 4, 2005. MIT, Cambridge, MA.

[2] Fox, and Pouly. An empirical investigation of mutation parameters and their effects on evolutionary convergence of a chess evaluation function. Lecture, May 9, 2005. MIT, Cambridge, MA.

[3] Misra, and Bednarz. Robust integrity monitoring. *GPS World* (April, 2004).